Boosting & Deep Learning

BOOSTING

- Ensemble Learning
- Bayes optimal Classifier
- Bagging
- Boosting
- Adaptive Boosting
- AdaBoost

Ensemble Learning

- So far – learning methods that learn a *single hypothesis*, chosen form a hypothesis space that is used to make predictions
- Ensemble learning → select a collection (ensemble) of hypotheses and combine their predictions
- Example
  - Generate 100 different decision trees from the same or different training set and have them vote on the best classification for a new example.
- Key motivation
  - Reduce the error rate. Hope is that it will become much more unlikely that the ensemble will misclassify an example.

Bayes optimal Classifier

A weighted majority classifier

- What is the most probable classification of the new **instance** given the training data?
  - The most probable classification of the new instance is obtained by combining the prediction of *all hypothesis*, weighted by their *posterior probabilities*
- If the classification of new example can take any value v_j from some set V, then the probability $P(v_j|D)$ that the correct classification for the new **instance is v_j**, is just *(remember, have to calculate $P(h_i|D)$)* :

$$P(v_j|D) = \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

Bayes optimal classification:

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

$$P(h_1|D) = .4, P(-|h_1) = 0, P(+|h_1) = 1$$

$$P(h_2|D) = .3, P(-|h_2) = 1, P(+|h_2) = 0$$

$$P(h_3|D) = .3, P(-|h_3) = 1, P(+|h_3) = 0$$

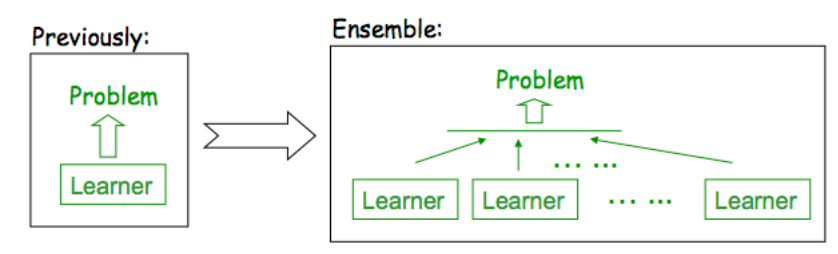
$$\sum_{h_i \in H} P(+|h_i)P(h_i|D) = .4 \qquad \sum_{h_i \in H} P(-|h_i)P(h_i|D) = .6$$

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = -$$

Gibbs Algorithm

- Bayes optimal classifier provides best result, but can be expensive if many hypotheses *(we have to compute the posterior probability for every hypothesis)*
- Gibbs algorithm:
  - Choose **one** hypothesis at random, according to *P(h|D)*
  - Use **this** to classify **new instance**
  - *Surprisingly its expected error no worse than twice Bayes optimal (Haussler et al. 1994)*

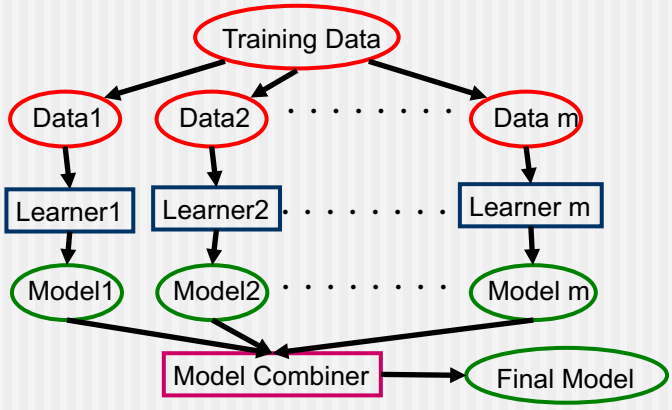
Ensemble Learning



- The generalization ability of the ensemble is usually significantly better than that of an individual learner
- Boosting is one of the most important families of ensemble methods

Learning Ensembles

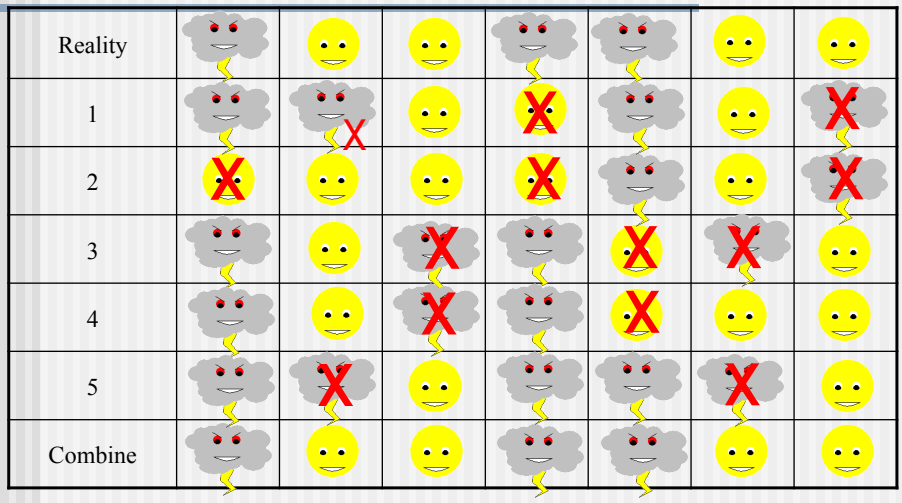
- Learn multiple alternative definitions of a concept using different training data or different learning algorithms.



Value of Ensembles

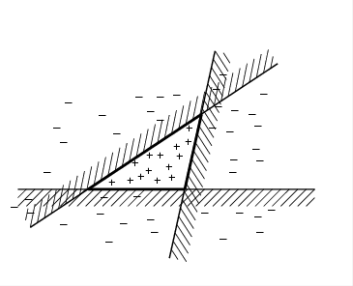
- "No Free Lunch" Theorem
  - No single algorithm wins all the time!
- When combing multiple **independent** and **diverse** decisions each of which is at least more accurate than random guessing, random errors cancel each other out, correct decisions are reinforced.
- Examples: Human ensembles are demonstrably better
  - How many jelly beans in the jar?: Individual estimates vs. group average.

Example: Weather Forecast



Ensemble Learning

- Ensemble learning:
- Way of enlarging the hypothesis space, i.e., the ensemble itself is a hypothesis and the new hypothesis space is the set of all possible ensembles constructible form hypotheses of the original space



Three linear threshold hypothesis (positive examples on the non-shaded side); Ensemble classifies as positive any example classified positively be all three.
The resulting triangular region hypothesis is not expressible in the original hypothesis space.

Different Learners

- Different learning algorithms
- Algorithms with different choice for parameters
- Data set with different features
- Data set = different subsets

Homogenous Ensembles

- Use a single, arbitrary learning algorithm but *manipulate training data* to make it learn multiple models.
  - *Data1 ≠ Data2 ≠ … ≠ Data m*
  - *Learner1 = Learner2 = … = Learner m*
- Different methods for changing training data:
  - Bagging: Resample training data
  - Boosting: Reweight training data

Bagging

- Create ensembles by repeatedly randomly resampling the training data (Brieman, 1996)
- Given a training set of size *n*, create *m* samples of size *n* by drawing *n* examples from the original data, ***with replacement***
  - Each ***bootstrap sample*** will on average contain 63.2% of the unique training examples, the rest are replicates
- Combine the *m* resulting models using simple majority vote

Bagging

- Combine the *m* resulting models using simple majority vote
- Decreases error by decreasing the variance in the results due to ***unstable learners***, algorithms (like decision trees) whose output can change dramatically when the training data is slightly changed

Bagging
Aggregate Bootstrapping

Given a standard training set *D* of size *n*

For i = *1 .. M*

Draw a sample of size $n^* < n$ from *D* uniformly and with replacement

Learn classifier C_i

Final classifier is a vote of $C_1 .. C_M$

Increases classifier stability/reduces variance

Strong and Weak Learners

- Strong Learner:
  - Objective of machine learning
  - Take labeled data for training
  - Produce a classifier which can be *arbitrarily accurate*
- Weak Learner
  - Take labeled data for training
  - Produce a classifier which is *more accurate than random guessing*

Boosting

- Weak Learner: only needs to generate a hypothesis with a training accuracy greater than 0.5, i.e., < 50% error over any distribution
- Learners
  - Strong learners are very difficult to construct
  - Constructing weaker Learners is relatively easy
- Questions: Can a set of **weak learners** create a single **strong learner** ?

Yes: Boost weak classifiers to a strong learner

Boosting

- Originally developed by computational learning theorists to guarantee performance improvements on fitting training data for a ***weak learner*** that only needs to generate a hypothesis with a training accuracy greater than *0.5* (Schapire, 1990).
- Revised to be a practical algorithm, AdaBoost, for building ensembles that empirically improves generalization performance (Freund & Shapire, 1996).

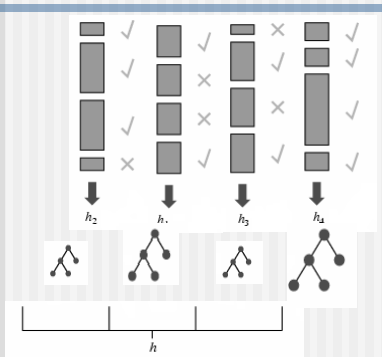
Boosting

- Examples are given weights
- At each iteration, a new hypothesis is learned and the examples are reweighted to focus the system on examples that the most recently learned classifier got wrong

Boosting

- Revised to be a practical algorithm, AdaBoost, for building ensembles that empirically improves generalization performance (Freund & Shapire, 1996).
- Instead of sampling (as in bagging) re-weigh examples!
- Examples are **given weights**.
  - At each iteration, a new hypothesis is learned (weak learner) and the *examples are reweighted* to focus the system on examples that the most recently learned classifier got wrong.
- Final classification based on **weighted vote of weak classifiers**

Adaptive Boosting



Each rectangle corresponds to an example, with weight proportional to its height.

Crosses correspond to *misclassified* examples.

Size of decision tree indicates *the weight of that hypothesis* in the final ensemble.

Construct Weak Classifiers

- Using Different Data Distribution
  - Start with **uniform weighting**
  - During each step of learning
    - *Increase weights* of the examples which are *not correctly learned* by the weak learner
    - *Decrease weights* of the examples which are *correctly learned* by the weak learner
- Idea
  - Focus on difficult examples which are not correctly classified in the previous steps

Combine Weak Classifiers

- Weighted Voting
  - Construct strong classifier by weighted voting of the weak classifiers
- Idea
  - **Better** weak classifier gets a **larger weight**
  - Iteratively add weak classifiers
    - Increase accuracy of the combined classifier through minimization of a cost function

Adaptive Boosting: High Level Description

C =0; /* counter*/

M = m; /* number of hypotheses to generate*/

1 Set same weight for all the examples *(typically each example has weight = 1*);

2 While (C < M)

2.1 Increase counter C by 1.

2.2 Generate hypothesis h_C .

2.3 Increase the weight of the misclassified examples in hypothesis h_C

3 Weighted majority combination of all M hypotheses (weights according to how well it performed on the training set).

- Many variants depending on how to set the weights and how to combine the hypotheses.
- ADABOOST → quite popular!!!!

Boosting: Basic Algorithm

- General Loop:

  Set all examples to have equal uniform weights.
  For t from 1 to T do:
  Learn a hypothesis, h_t, from the weighted examples
  Decrease the weights of examples h_t classifies correctly
- Base (weak) learner must focus on correctly classifying the most highly weighted examples while strongly avoiding over-fitting.
- During testing, each of the T hypotheses get a weighted vote proportional to their accuracy on the training data.

AdaBoost Pseudocode

TrainAdaBoost(D, BaseLearn)
{
For each example d_i in D let its weight w_i=1/|D|
Let H be an empty set of hypotheses
For t from 1 to T do:
{
Learn a hypothesis, h_t, from the weighted examples: h_t=BaseLearn(D)
Add h_t to H
Calculate the error, ε_t, of the hypothesis h_t as the total sum weight of the examples that it classifies incorrectly.
If $\varepsilon_t > 0.5$ then T:=t-1 and recommence loop
Let $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$ (or $\beta_t = \frac{1}{2} \log(\varepsilon_t / (1 - \varepsilon_t))$)
Multiply the weights of the examples that h_t classifies correctly by β_t
Rescale the weights of all of the examples so the total sum weight remains 1.
}
Return H
}

AdaBoost Pseudocode

TestAdaBoost(*ex*, *H*)
{
Let each hypothesis, h_t, in *H* vote for *ex*' s classification with weight log(1/ β_t)
Return the class with the highest weighted vote total.
}

Performance of Adaboost

- Learner = Hypothesis = Classifier
- Weak Learner: < 50% error over any distribution
- *M* number of hypothesis in the ensemble.
- If the input learning is a Weak Learner, then ADABOOST will return a hypothesis that classifies the training data perfectly for a large enough *M*, boosting the accuracy of the original learning algorithm on the training Data.
- Strong Classifier: thresholded linear combination of weak learner outputs.

Experimental Results on Ensembles
(Freund & Schapire, 1996; Quinlan, 1996)

- Ensembles have been used to improve generalization accuracy on a wide variety of problems.
- On average**, Boosting** provides a larger increase in accuracy than **Bagging**.
- Boosting on rare occasions can degrade accuracy.
- Boosting is particularly subject to over-fitting when there is significant noise in the training data.

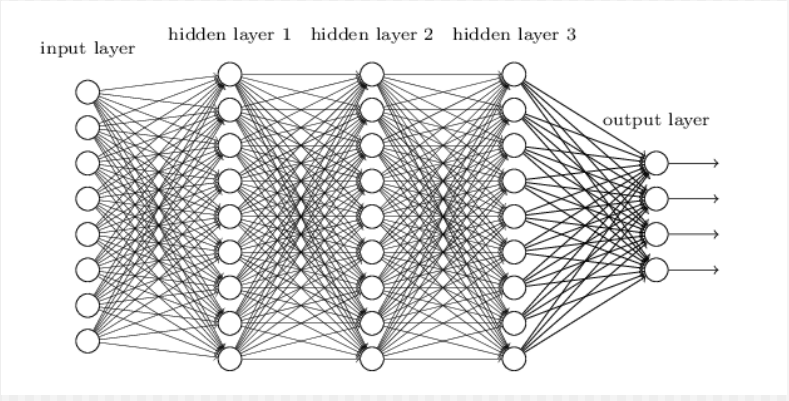
DEEP LEARNING

- Deep Learning and Backpropagation
- Vanishing gradients
- Convolutional Neural Networks
- Brain and Deep Learning
- HMAX
- Neocognitron
- Map Transformation Cascade

Deep Learning and Backpropagation

- According to the universality theorem, a neural network with a single hidden layer is capable of approximating any continuous function
- However, attempting to build a network with only one layer to approximate complex functions often requires a very large number of nodes

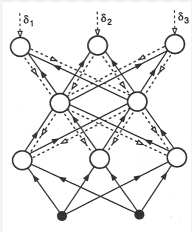
- The immediate solution to this is to build networks with more hidden layers



- Many hidden layers and the resulting problems:
  - (a) Vanishing gradients
  - (b) Overfitting

(a) Vanishing gradients

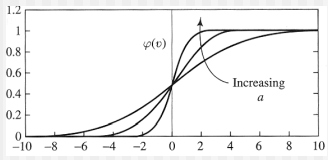
- Backpropagation
- Error gradient reaches a layer with nodes in saturation (i.e, with outputs close to 1 and derivatives close to 0).



- The gradient is back propagated to the next, lower-level layer as a very small value, and approaches zero as it reaches the input layer
- Weight updates in lower-level layers are insignificant

- Impractical to train networks with many layers when using sigmoid or hyperbolic tangent activations

$$f(x) = \sigma(x) = \frac{1}{1 + e^{(-\alpha \cdot x)}}$$



$$f'(x) = \sigma'(x) = \alpha \cdot \sigma(x) \cdot (1 - \sigma(x))$$

$$f(x) = \tanh(\alpha \cdot x)$$

$$f'(x) = \alpha \cdot (1 - f(x)^2)$$

- A solution to the vanishing gradients came with the introduction of the Rectified Linear Activation Function

$$f(x) = max(0, x).$$

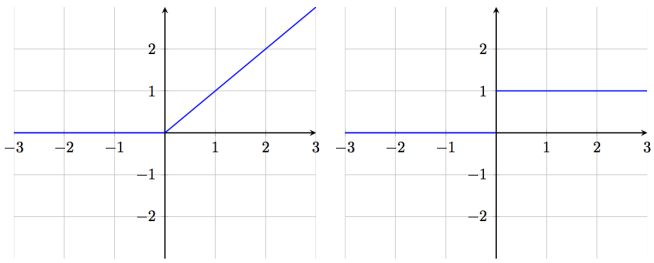


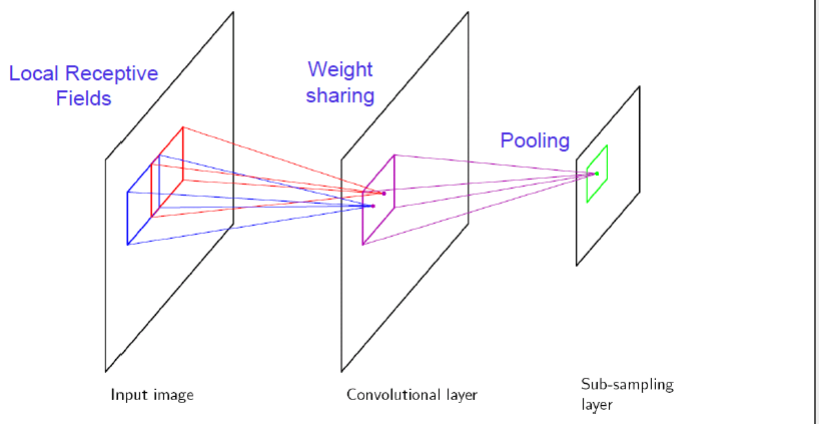
Figure 2.7: The ReLU function (left) and its derivative (right).

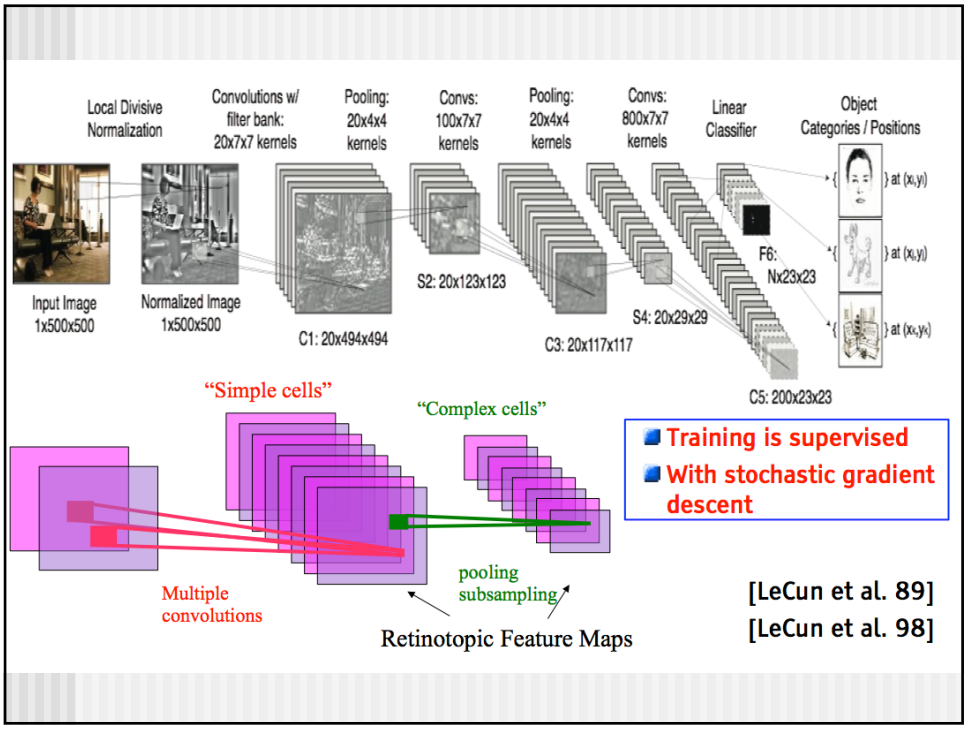
(b) Overfitting

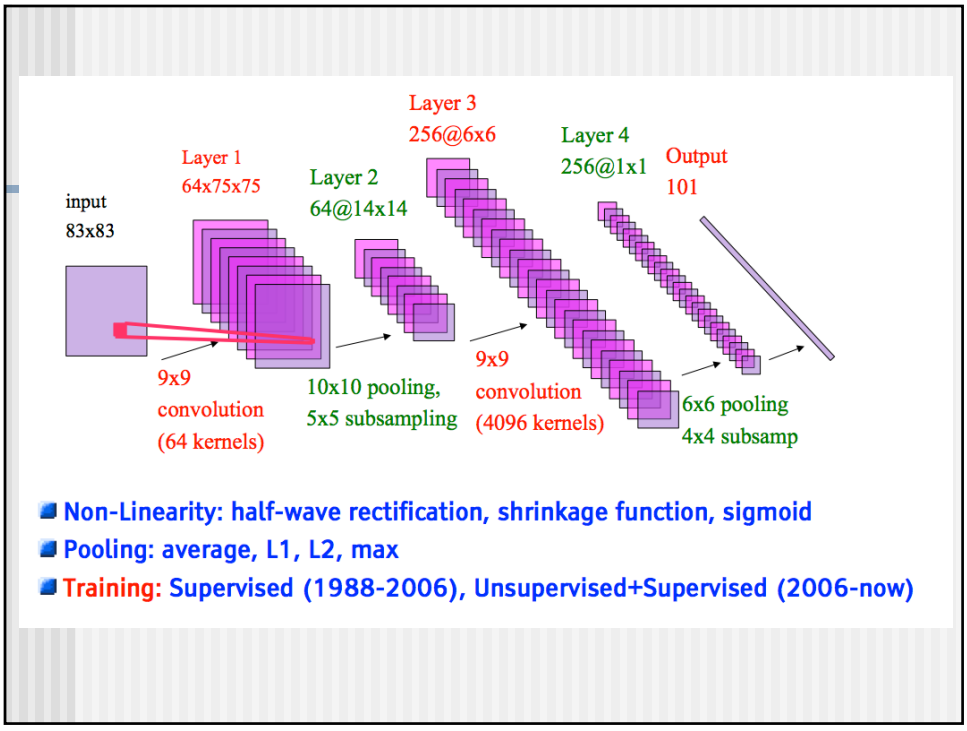
- Solution, reduce the number of weights
- Cut weights randomly
- Represent the topology of the input patterns

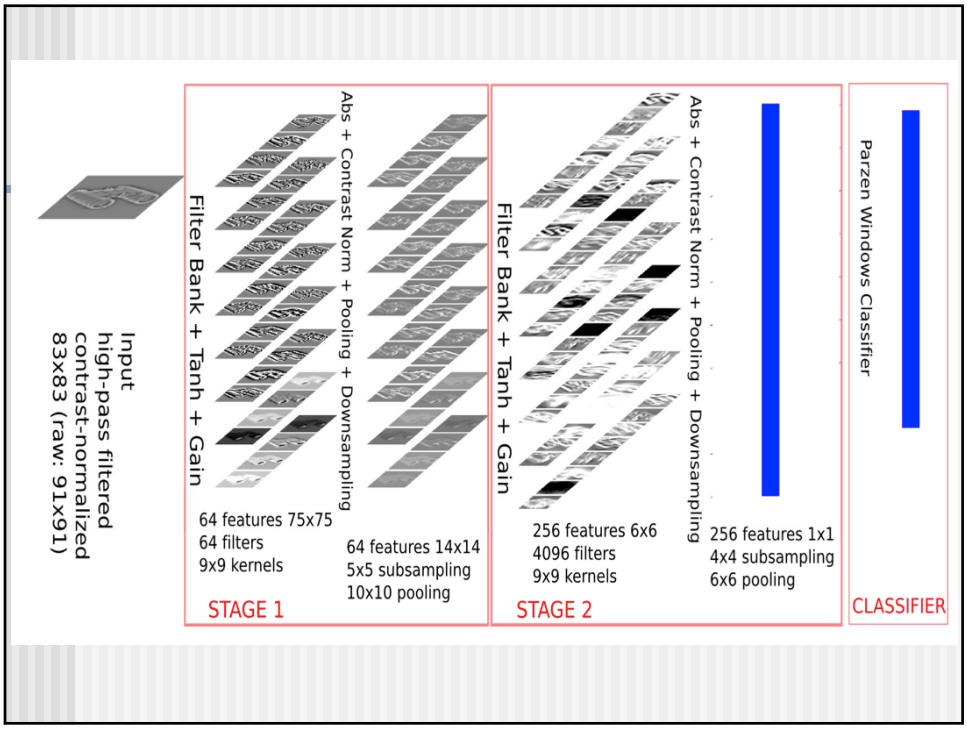
Convolutional Neural Networks

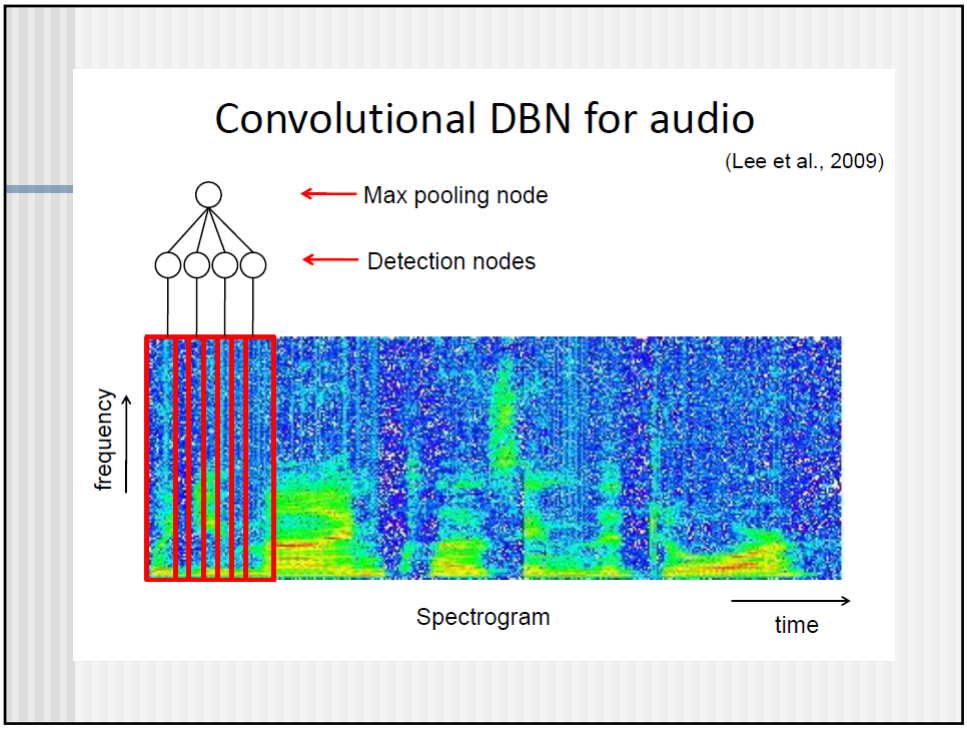
(LeCun et al., 1989)



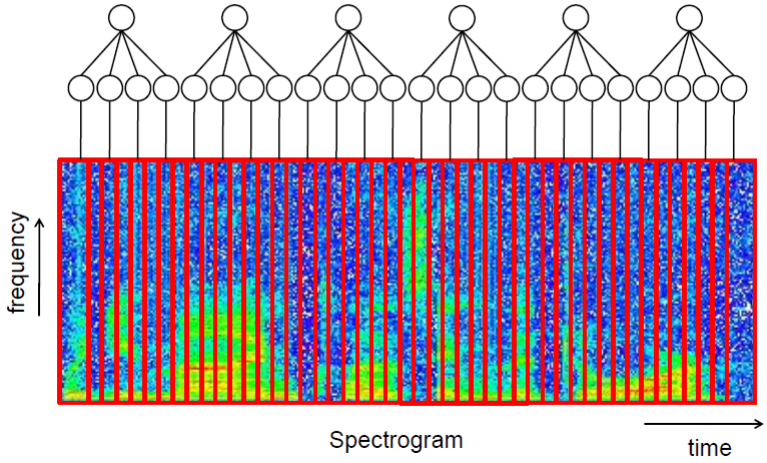

Local Divisive Normalization
Convolutions w/ filter bank: 20x7x7 kernels
Pooling: 20x4x4 kernels
Convs: 100x7x7 kernels
Pooling: 20x4x4 kernels
Convs: 800x7x7 kernels
Linear Classifier
Object Categories / Positions
Input Image 1x500x500
Normalized Image 1x500x500
C1: 20x494x494
S2: 20x123x123
C3: 20x117x117
S4: 20x29x29
F6: Nx23x23
C5: 200x23x23
} at (xi,yi)
} at (xj,yj)
} at (xk,yk)
"Simple cells"
"Complex cells"
Multiple convolutions
pooling subsampling
Retinotopic Feature Maps
Training is supervised
With stochastic gradient descent
[LeCun et al. 89]
[LeCun et al. 98]


Layer 3
256@6x6
Layer 4
256@1x1
Output
101
Layer 1
64x75x75
Layer 2
64@14x14
input
83x83
9x9
convolution
(64 kernels)
10x10 pooling,
5x5 subsampling
9x9
convolution
(4096 kernels)
6x6 pooling
4x4 subsamp
Non-Linearity: half-wave rectification, shrinkage function, sigmoid
Pooling: average, L1, L2, max
Training: Supervised (1988-2006), Unsupervised+Supervised (2006-now)


Input
high-pass filtered
contrast-normalized
83x83 (raw: 91x91)
Filter Bank + Tanh + Gain
Abs + Contrast Norm + Pooling + Downsampling
64 features 75x75
64 filters
9x9 kernels
64 features 14x14
5x5 subsampling
10x10 pooling
STAGE 1
Filter Bank + Tanh + Gain
Abs + Contrast Norm + Pooling + Downsampling
256 features 6x6
4096 filters
9x9 kernels
256 features 1x1
4x4 subsampling
6x6 pooling
STAGE 2
Parzen Windows Classifier
CLASSIFIER


Convolutional DBN for audio
(Lee et al., 2009)
Max pooling node
Detection nodes
frequency
Spectrogram
time

Convolutional DBN for audio



Deep Learning beyond backpropagation

- Deep Architectures can be representationally efficient
  - Fewer computational units for same function
  - Deep Representations might allow for a hierarchy
    Comprehensibility
- Multiple levels of latent variables allow combinatorial sharing of statistical strength

Deep Network Training

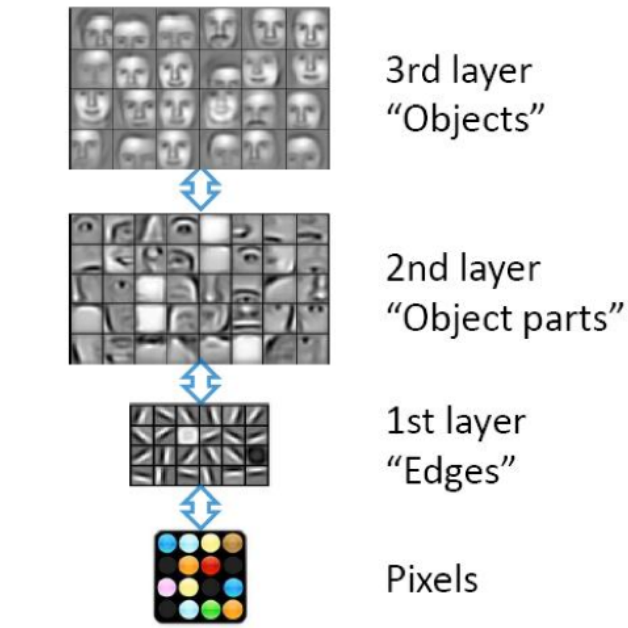
- Use unsupervised learning (greedy layer-wise training)
  - Allows abstraction to develop naturally from one layer to another
  - Help the network initialize with good parameters
- Perform supervised top-down training as final step
  - Refine the features (intermediate layers) so that they become more relevant for the task

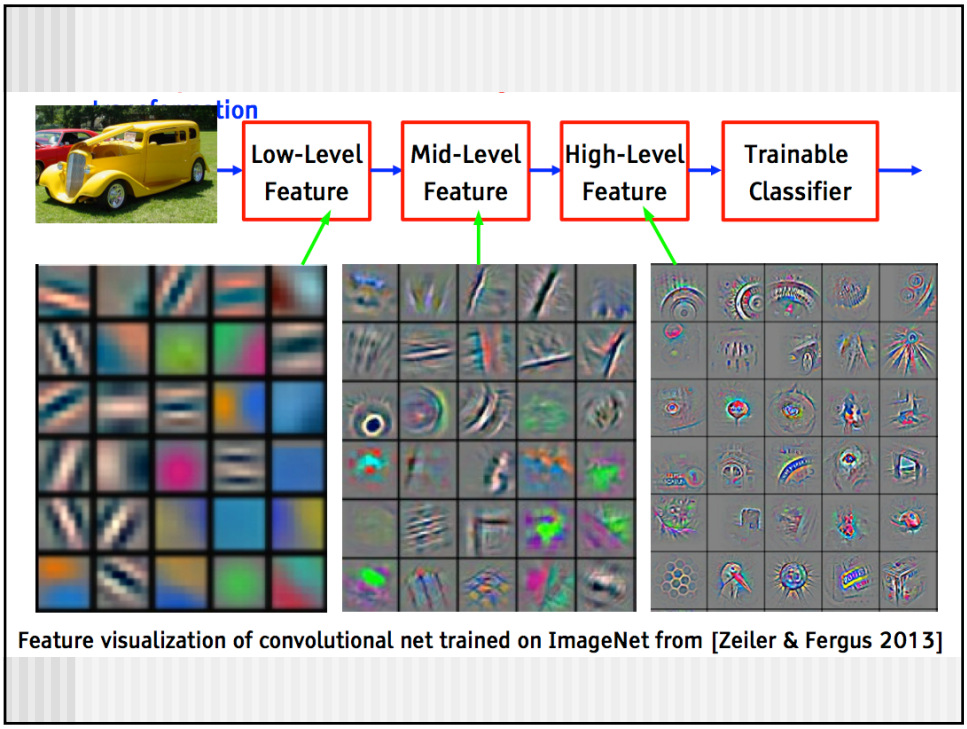
Different Levels of Abstraction

- Hierarchical Learning
- Natural progression from low level to high level structure as seen in natural complexity
- Easier to monitor what is being learnt and to guide the machine to better subspaces
- A good lower level representation can be used for many distinct tasks

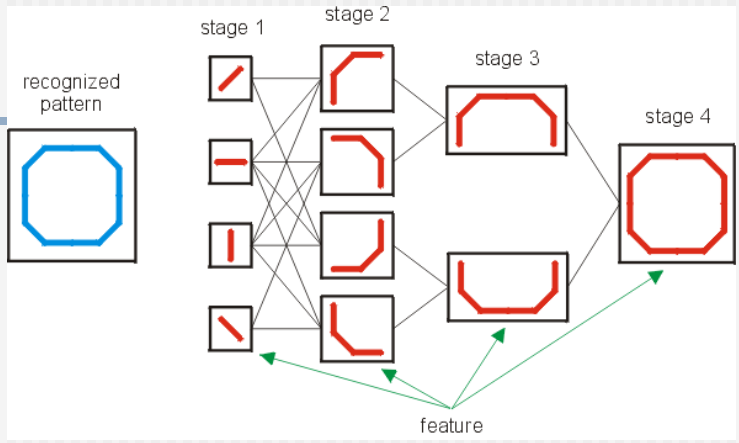
Hierarchy of representations with increasing level of abstraction

- Each stage is a kind of trainable feature transform
  - Image recognition
    - Pixel → edge → texton → motif → part → object
  - Text
    - Character → word → word group → clause → sentence → story
  - Speech
    - Sample → spectral band → sound → ... → phone → phoneme → word →





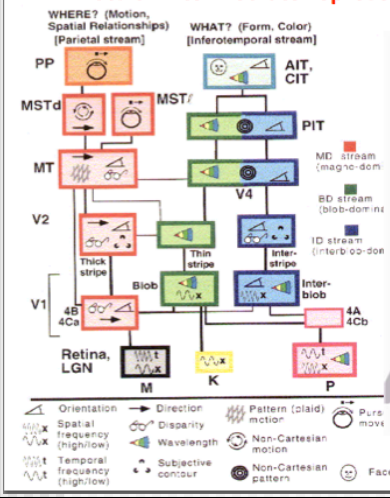
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

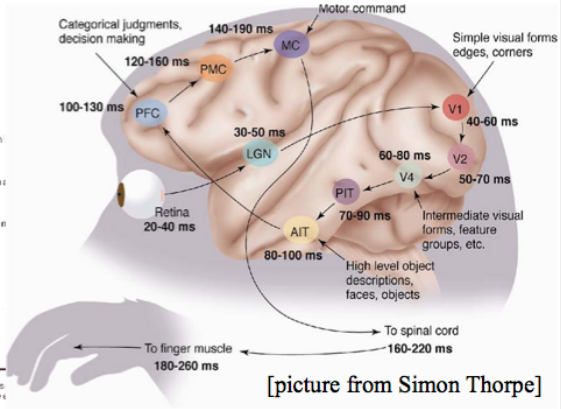


- Image passed through layers of units with progressively more complex features at progressively less specific locations.
- Hierarchical in that features at one stage are built from features at earlier stages

Brain and Deep Learning

- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations

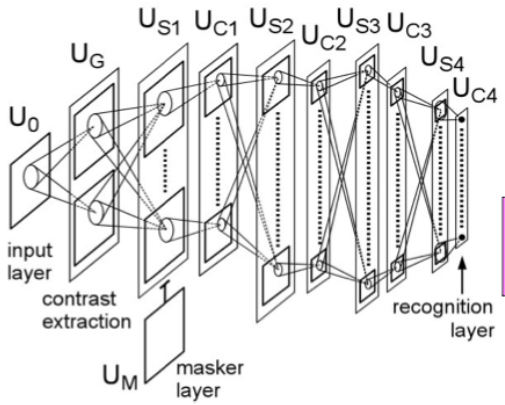


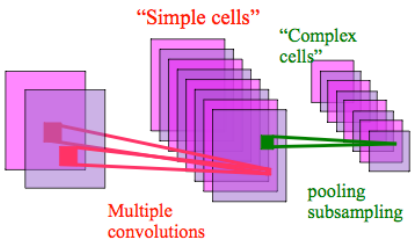


[picture from Simon Thorpe]

[Gallant & Van Essen]

- [Hubel & Wiesel 1962]:
  - simple cells detect local features
  - complex cells "pool" the outputs of simple cells within a retinotopic neighborhood.





Cognitron & Neocognitron [Fukushima 1974-1982]

- Retina translates light into nerve signals
- Descriminates wavelength so we can see colors
- Retina is connected to the rest of the brain trough the optic nerve
- Retina's output is produced by the ganglion cells' outputs which bundle in the optic nerve
- Two main types of ganglion cells
  - on-center and off-center

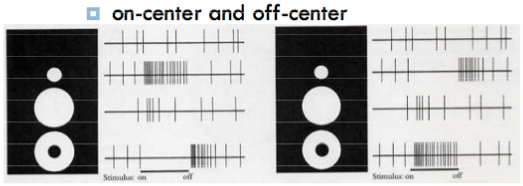


Fig. – Left: on-center ganglion cells response and Right: off-center

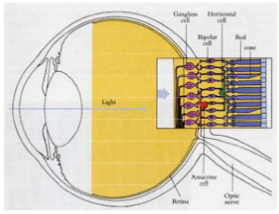


Fig. – A cross section detail of the retina

- Receptive field refers to an area on which the response of a cell depends and how that area should be stimulated to get a response.
  - Deeper into the visual cortex these descriptions become more complex
- Neighbour ganglion cells have overlapping receptive fields
  - Topographical organization is also present throughout the visual cortex

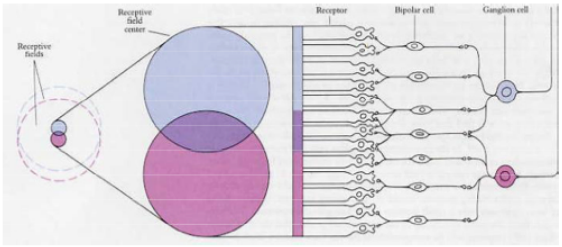
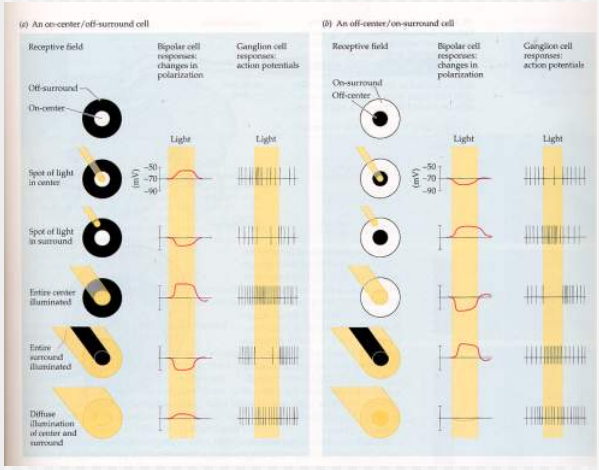


Fig. – Retinal Ganglion Cells Receptive Fields

Receptive Fields in Retinal Bipolar and Ganglion Cells


Stephen Kuffler
(1913–1980)

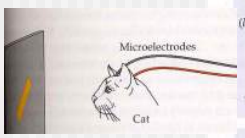
Horace Barlow

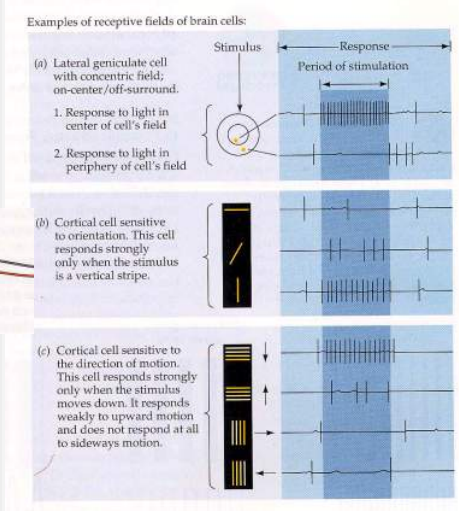


Receptive Fields of Lateral Geniculate and Primary Visual Cortex


David Hubel


Torsten Wiesel





- Simple Cells
  - Recognize patterns with a specific orientation and position
  - Their receptive field size depends specially on their position regarding the retina area to which they react
    - Even for a particular part of the retina their size can vary

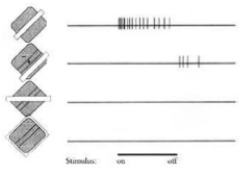


Fig. – Simple Cell Response

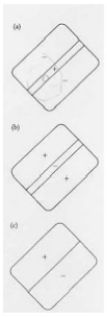


Fig. – Three typical receptive fields for simple cells

- Complex Cells
  - Recognize patterns with a specific orientation but allow a positional shift.
  - Most common type of cells in the primary visual cortex
  - Some exhibit directional selectivity

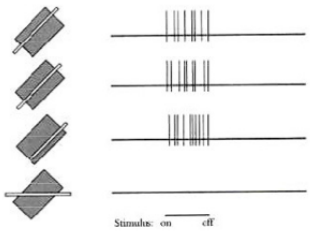


Fig. – Complex Cell

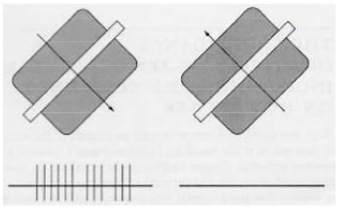


Fig. – Complex Cell with Directional Selectivity

- The visual cortex is composed essentially as an hierarchy of cells
  - Layers of simple and complex cells are arranged in a hierachical way
  - The input of a layer is the output of the previous layer

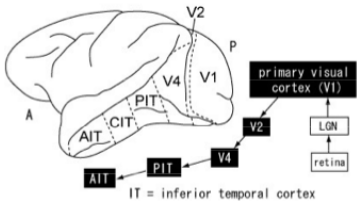


Fig. – Visual pathway [nips.ac.jp]

- Throughout the visual cortex there is a gradual increase in the complexity of the preferred stimulus
- The receptive field sizes and invariance properties also increase gradually

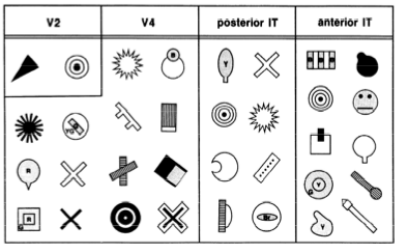


Fig. – Increasing Complexity in prefered stimulus [Kobatake et al. 94]

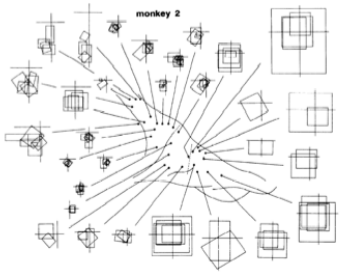


Fig. – Receptive fields from a region including V4 and IT [Kobatake et al. 94]

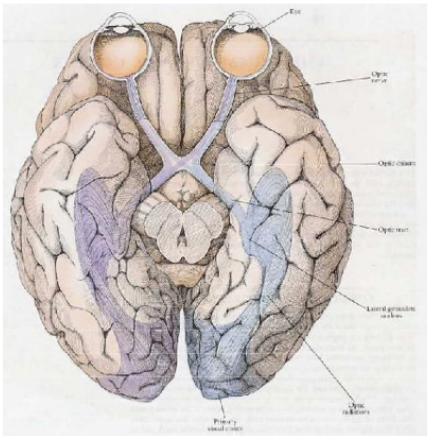


Fig. – Primary visual cortex location in humans

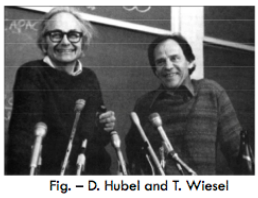


Fig. – D. Hubel and T. Wiesel
1981 Nobel Prize in Physiology or Medicine
For their discoveries concerning information processing in the visual system

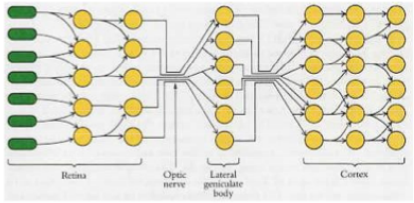
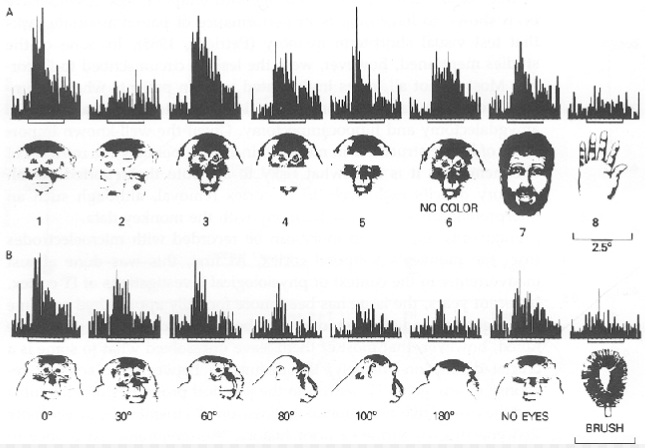


Fig. – Visual processing pathway

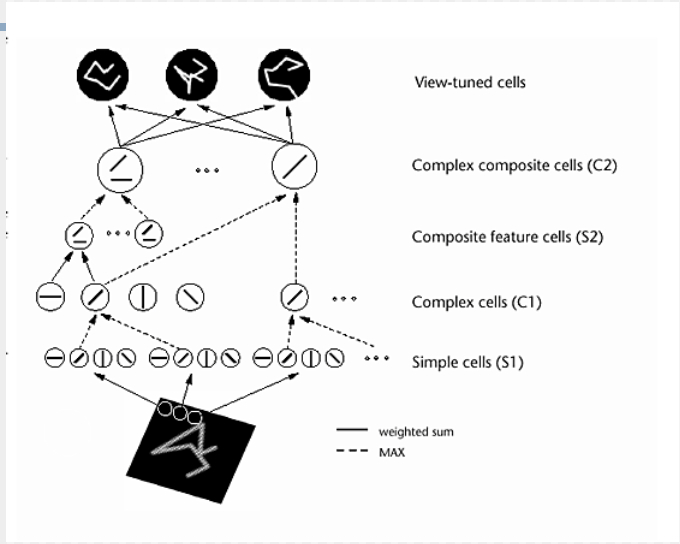
*Section figures from "Eye, Brain and Vision" by D. Hubel

Face Cells in Monkey



Computational Model of Object Recognition - HMAX

(Riesenhuber and Poggio, 1999)



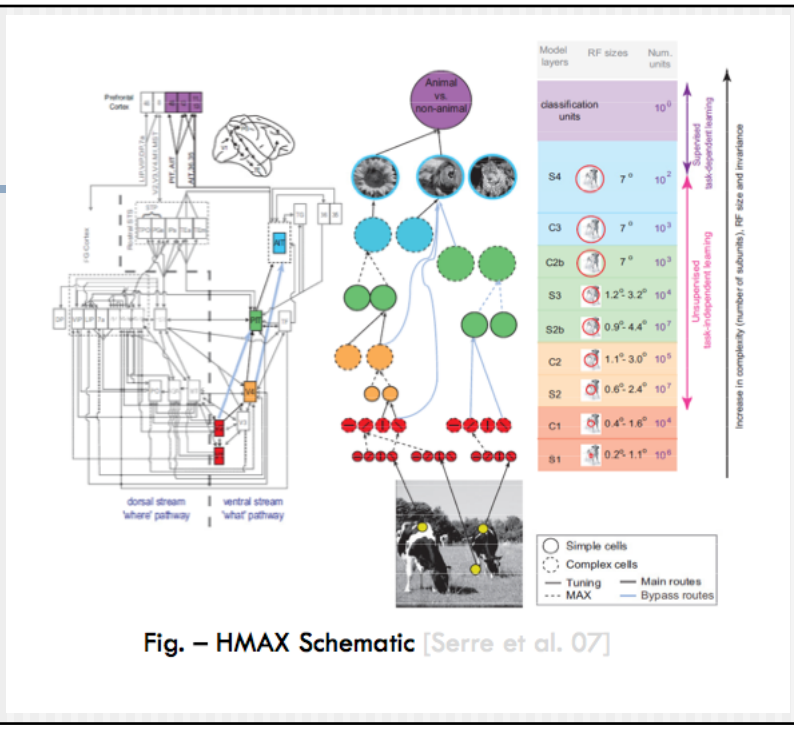
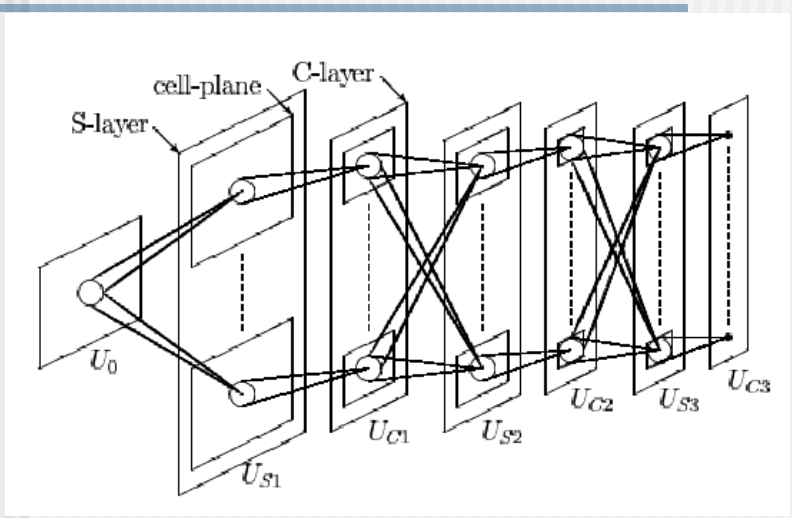
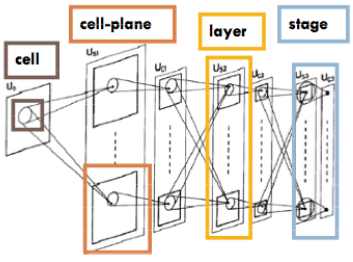


Fig. – HMAX Schematic [Serre et al. 07]

Fukushima & Miyake (1982)'s **Neocognitron**



- Cell
  - represents a biological cell, has specific number of connections in a certain position
- Cell-plane
  - a group of cells of the same type, which all recognize the same feature, like a feature map
- Layer
  - set of cell-planes of the same type of cells
- Stage
  - an ordered pair of layers in which the first is an S-cell Layer (simple) and the second a C-cell Layer (complex)

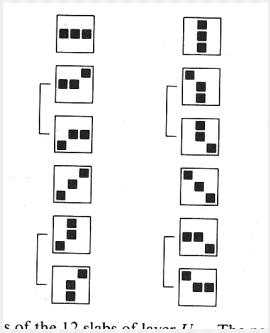


adapted from [Fukushima 03]

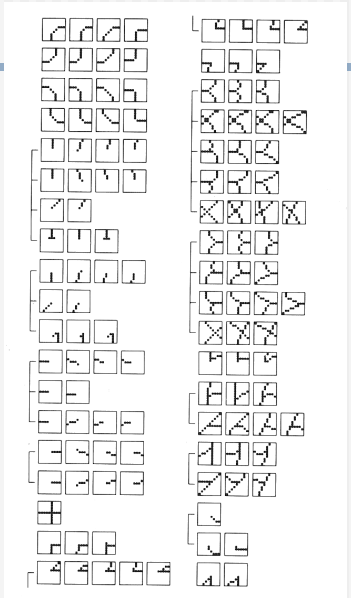
- S-cells
  - represent simple cells in the visual cortex
    - Extract features
  - Learn to form a template of particular feature in particular position
  - Share a weight-vector with all cells in their cell-plane
    - In a cell-plane all cells extract the same feature in different positions
- C-cells
  - Represent complex cells in the visual cortex
    - Allow positional shifts in features
  - It's output is a blurred version of their input

- Sequential Learning
  - Each stage is trained separately
    - S-cells connections are changed by learning
    - C-cells connections are fixed
    - Starting from the stages close the input layer
    - The training of a stage only starts after all the preceding one's is finished
- Unsupervised learning
  - Except the final stage which corresponds to a classifier
    - Assign a label to each cell-plane in the last stage
      - if the winner cell has the same label as the input stimulus then reinforce connections
      - If the winner cell has a different label from the input stimulus create a new cell-plane for the presented input stimulus
- Other variations exist

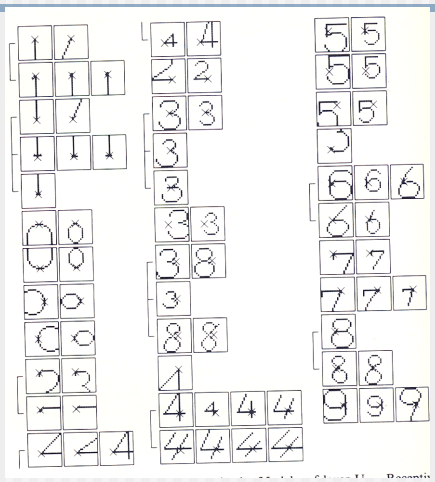
First S-layer after learning



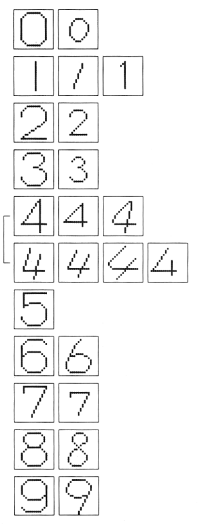
Second S-layer

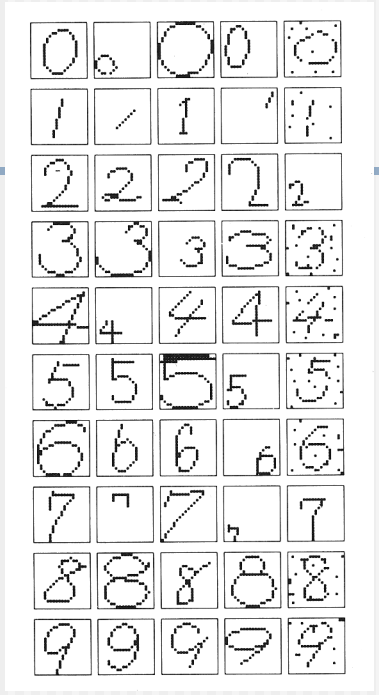


Third S-layer

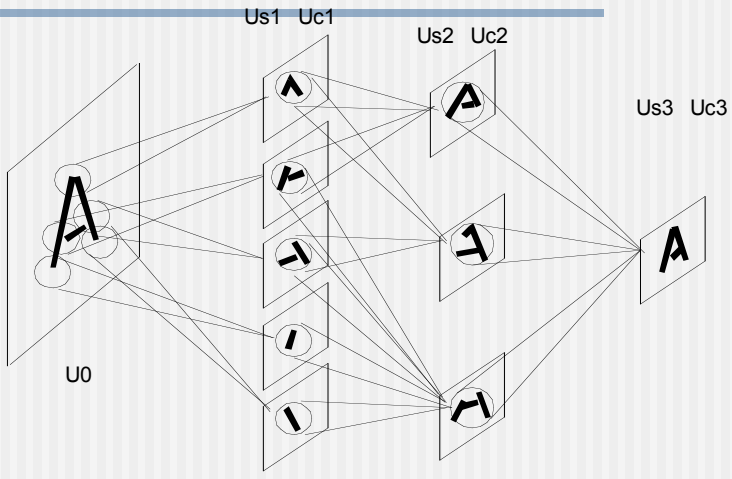


Fourth S-cell layer





- Exemplo de fatores reconhecidos pelos neurônios nos 3 estágios de uma implementação do neocognitron.



- C-cells resemble complex cells in the visual cortex
- Their purpose is to allow positional changes and distortions of the features
- They do this by blurring the stimulus they receive

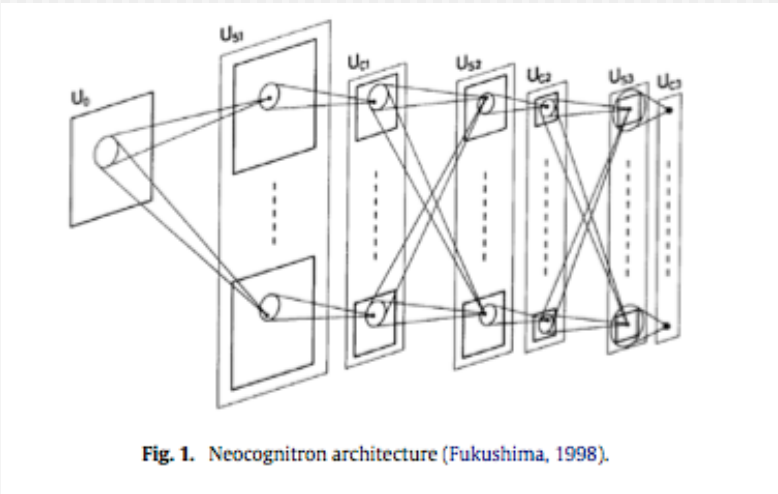
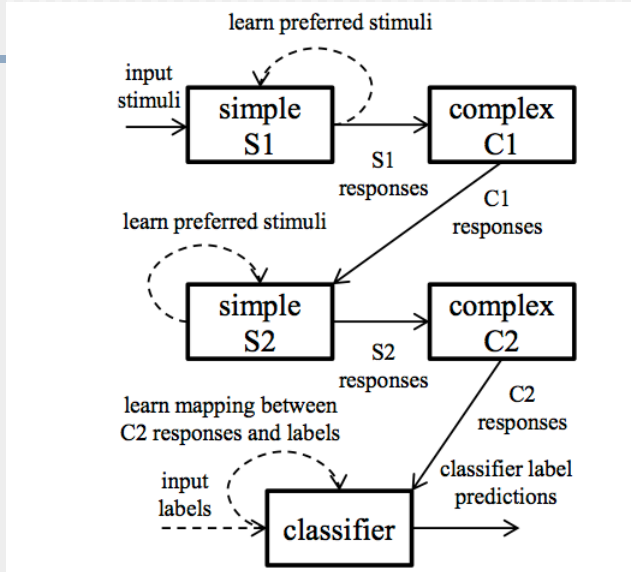
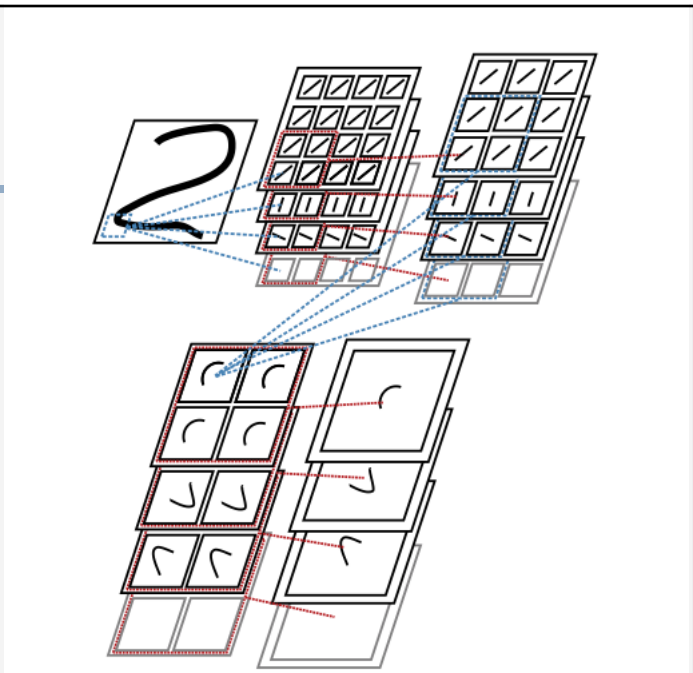
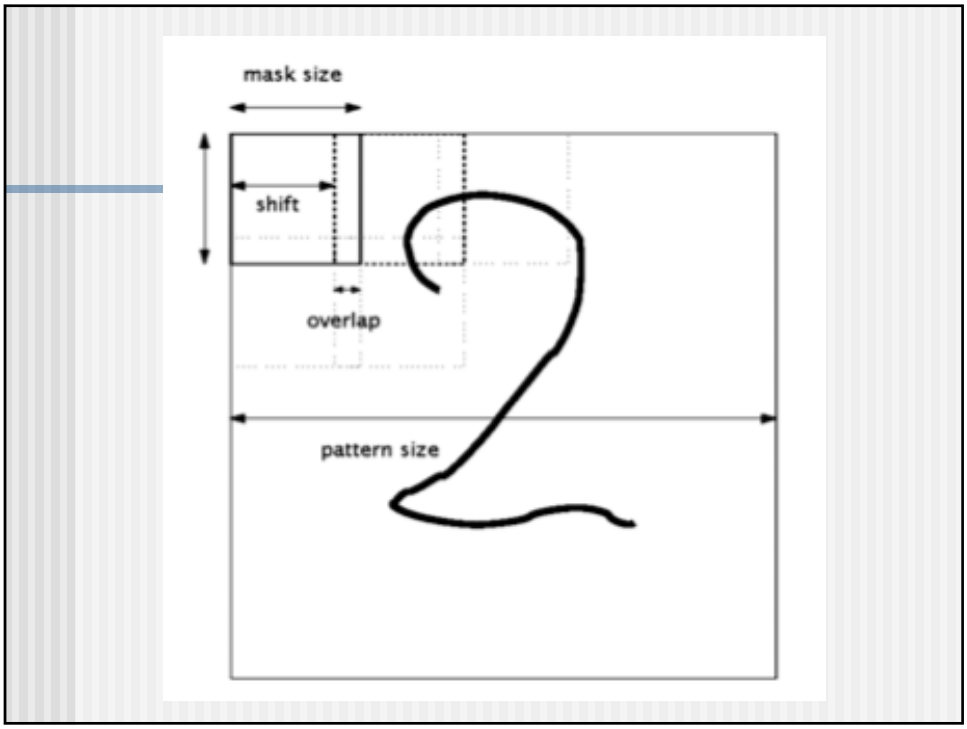


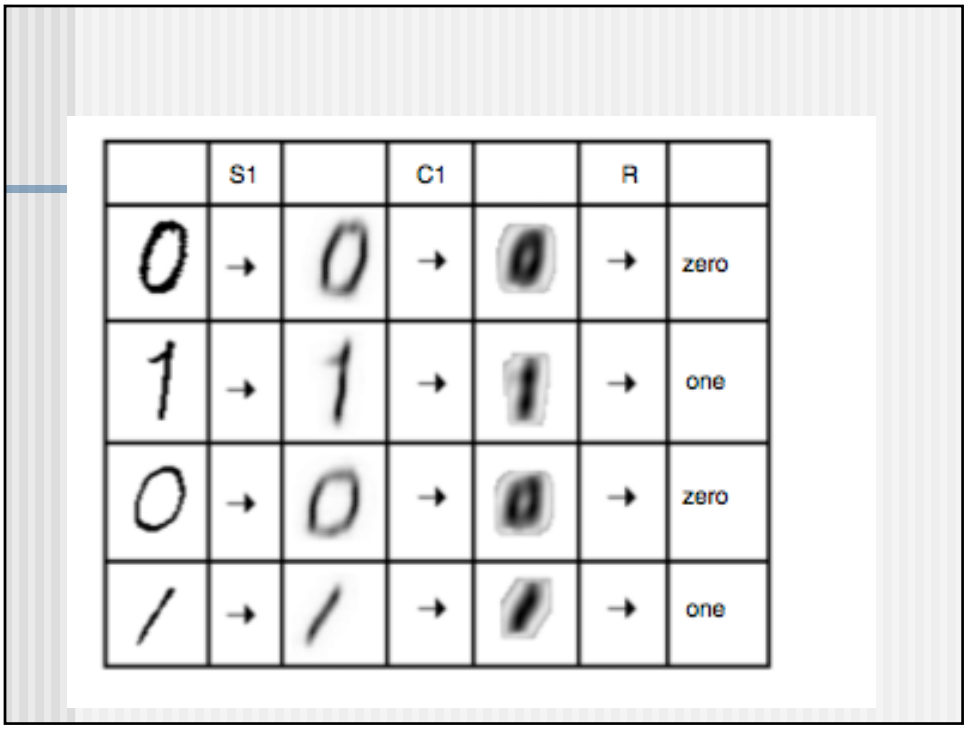
Fig. 1. Neocognitron architecture (Fukushima, 1998).

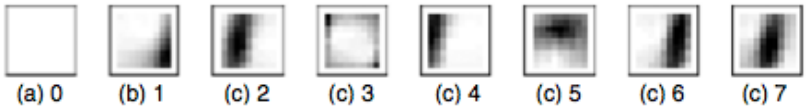
Map Transformation Cascade






mask size
shift
overlap
pattern size


S1
C1
R
zero
one
zero
one



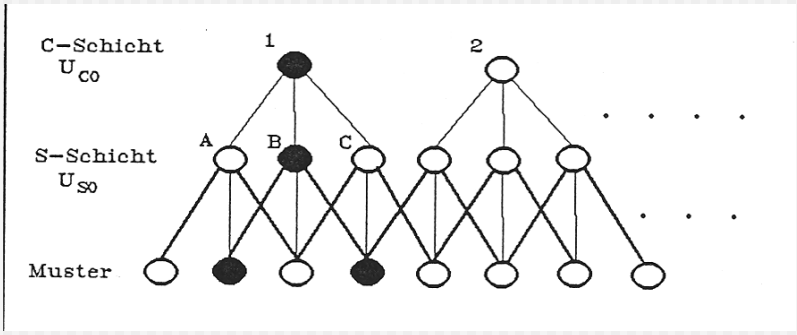
(a) 0 (b) 1 (c) 2 (c) 3 (c) 4 (c) 5 (c) 6 (c) 7

(a) Output: 7. (b) Output: 2. (c) Output: 4. (d) Output: 3.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	3	3	1	1	3	3	3	0	0	0	0
0	0	0	0	0	3	1	1	1	1	1	1	3	3	3	0	0	0
0	0	0	0	3	1	1	1	1	5	5	1	7	2	3	3	0	0
0	0	0	3	1	1	7	3	5	5	5	5	5	2	4	3	0	0
0	0	3	1	1	7	5	5	5	5	5	5	5	7	2	4	0	0
0	0	3	1	6	7	2	4	3	3	3	1	6	7	2	4	3	0
0	0	1	6	7	2	4	3	0	0	0	3	6	7	2	4	3	0
0	3	1	6	7	2	4	3	0	0	0	3	6	7	2	4	3	0
0	3	1	7	2	4	3	0	0	0	0	3	6	7	2	4	3	0
0	1	6	7	2	4	3	0	0	0	0	1	6	7	2	4	3	0
0	1	6	7	2	4	0	0	0	0	0	1	6	7	2	4	3	0
0	1	6	7	2	4	0	0	0	0	3	1	6	7	4	4	0	0
0	1	6	7	2	3	0	0	0	3	1	6	7	2	4	3	0	0
0	1	6	7	2	4	3	3	3	1	1	7	7	2	4	3	0	0
0	1	6	7	2	4	1	1	1	1	7	7	2	4	3	0	0	0
0	3	6	5	5	1	1	1	7	7	5	5	3	3	3	0	0	0
0	3	3	5	5	5	5	5	5	5	5	5	3	3	3	0	0	0
0	0	3	3	5	5	5	5	5	4	3	3	0	0	0	0	0	0
0	0	0	3	3	3	3	3	3	3	3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	1	6
0	0	0	3	6
0	0	0	3	3
0	0	0	0	3
0	0	0	0	0

Fig. 8. C-Layer mask input in a given position when scanning Fig. 7, its output is {1, 6, 3}. It indicates the presence of these classes.



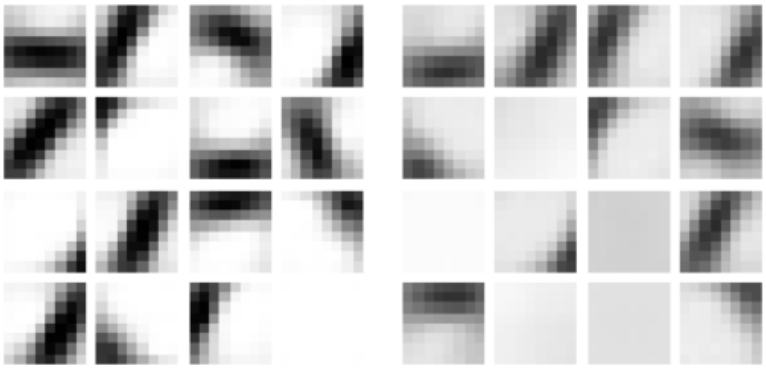
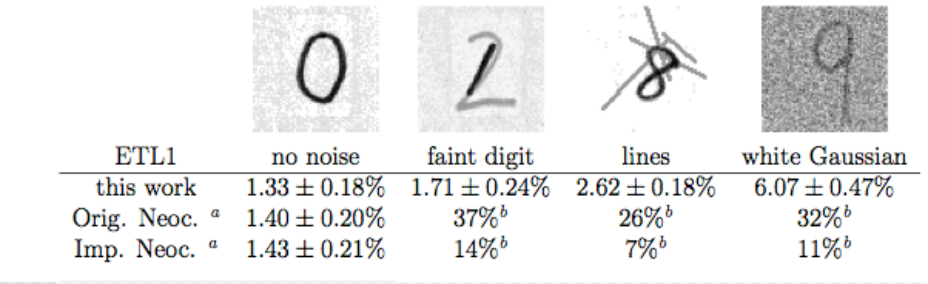


Figure 4.1: A set of 16 preferences learned using K-means on ETL1 for 100000 patches of size 8×8. On the left, the preferences using binary versions of the patterns. On the right, the preferences using grayscale versions. We can see that the grayscale versions of the patterns produce low-contrast preferences. Several of the preferences are simply different shades of gray.



ETL1	no noise	faint digit	lines	white Gaussian
this work	$1.33 \pm 0.18\%$	$1.71 \pm 0.24\%$	$2.62 \pm 0.18\%$	$6.07 \pm 0.47\%$
Orig. Neoc. [a]	$1.40 \pm 0.20\%$	$37\%^{b}$	$26\%^{b}$	$32\%^{b}$
Imp. Neoc. [a]	$1.43 \pm 0.21\%$	$14\%^{b}$	$7\%^{b}$	$11\%^{b}$

Map Transformation Cascade

- Wichert, A.: MTCn-Nets. Proceeding World Congres on Neural Networks 1993, Vol.IV, pp.59-62, Lawrence Erlbaum, 1993
- [Kemke C.,Wichert A.: Hierarchical Self-Organizing Feature Maps for Speech Recognition. Proceedings World Congress on Neural Networks 1993, Vol. III, pp.45-47, Lawrence Erlbaum, 1993
- Cardoso, A, Wichert A.: Neocognitron and the Map Transformation Cascade, Neural Networks, 23 (1): 74-88, 2010 doi:10.1016/j.neunet.2009.09.004
- Angelo Cardoso and Andreas Wichert: Handwritten digit recognition using biologically inspired features, Neurocomputing, 99: 575-589, 2013 doi:10.1016/j.neucom.2012.07.027
- Angelo Cardoso and Andreas Wichert, Noise tolerance in a Neocognitron like network, Neural Networks, 49: 32-28, 2014 doi:10.1016/j.neunet.2013.09.007